BAB IV

IMPLEMENTASI DAN PEMBAHASAN

4.1 Implementasi dan Uji Coba Sistem

Implementasi program berdasarakan rancangan sistem yang dibuat meliputi.

4.1.1 Pemetaan Lokasi *Dealer* Motor *Yamaha*

API yang digunakan dalam sistem adalah M*apBox*. Untuk menampilkan peta kedalam aplikasi *android* kode yang digunakan adalah.

a) Menampilkan *MapView* kedalam *layout* aplikasi *android*. Kode XML untuk menampilkan *MapView* dengan *mapbox-sdk* yaitu.

```
<com.mapbox.mapboxsdk.maps.MapView
android:id="@+id/mapview"
android:layout_width="match_parent"
android:layout_height="match_parent"
mapbox:mapbox_styleUrl="@string/mapbox_style_mapbox_streets"
mapbox:mapbox_cameraZoom="10"
mapbox:mapbox_cameraTargetLat="-7.712622"
mapbox:mapbox_cameraTargetLng="110.592131"/>
```

Gambar 4.1. Komponen *MapBox*

b) Membuat variable yang digunakan untuk membuat variabel dan memanipulasi *MapView* yaitu.

```
mapView = (MapView) findViewById(R.id.mapview);
mapView.onCreate(savedInstanceState);
mapView.getMapAsync(new OnMapReadyCallback() {
   @Override
    public void onMapReady(MapboxMap mapboxMap) {
        map = mapboxMap;
});
```

Gambar 4.2 Variabel untuk Mapview

Hasil luaran implementasi dari contoh kode program pada gambar 4.1 dan gambar 4.2 digambarkan pada *screen shoot* gambar 4.8

Gambar 4.3 Implementasi komponen *mapbox-sdk*

4.1.2 Visualisasi *Marker* Berdasarkan *Rating*

Dalam memvisualisasikan *marker* pada *mapview*, *dealer* dengan nilai *rating* tertinggi akan ditampilkan dengan *marker* yang paling besar dan *dealer* dengan *rating* rendah akan ditampilkan dengan *marker* yang lebih kecil. Perbandingan *marker dealer* yang diterapkan pada *mapview* dipaparkan pada tabel 4.1.

Tabel 4.1 Perbandingan *Marker* Berdasarkan *Rating Dealer*

Rating	**Ukuran *marker***
Null (belum ada rating) atau 0 s/d 1	24 x 32 pixel (32 bit color) PNG
1 s/d 2	30 x 40 pixel (32 bit color) PNG
2 s/d 3	36 x 49 pixel (32 bit color) PNG
3 s/d 4	42 x 56 pixel (32 bit color) PNG
4 s/d 5	49 x 65 pixel (32 bit color) PNG

Perbandigan ukuran *marker* pada tabel 4.1 diimplementasikan pada pada program. Kode seleksi ukuran *marker* berdasarkan *rating* dipaparkan pada gambar 4.4

```
private Icon getIcon(String nil){
  double nilai;
  if (!nil.equals("null") ){nilai=Double.parseDouble(nil); }
  else {nilai = 0.0; }
  if(nilai < 1){
    icon = iconFactory.fromResource(R.drawable.marker);
  }else if(nilai < 2){
    icon = iconFactory.fromResource(R.drawable.marker2);
  }else if(nilai < 3){
    icon = iconFactory.fromResource(R.drawable.marker3);
  }else if(nilai < 4){
    icon = iconFactory.fromResource(R.drawable.marker4);
  }else
    icon = iconFactory.fromResource(R.drawable.marker5);
        return icon;
}
```

Gambar 4.4 Kode penentuan ukuran *icon marker*

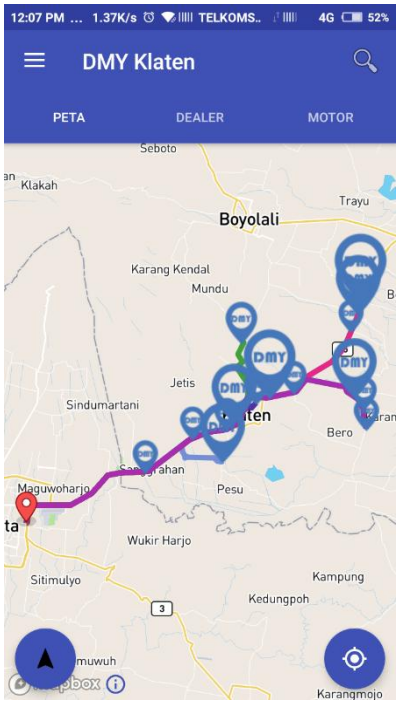
4.1.3 Visualisasi Rute *Dealer*

Untuk membuat visualisasi rute *dealer*, kode program yang digunakan dipaparkan dalam gambar 4.5.

```
fabAllRute.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
for (int i = 0; i < memberList.size(); i++) {

double lat =
Double.parseDouble(memberList.get(i).get(Config.LATITUDE));
double lng =
Double.parseDouble(memberList.get(i).get(Config.LONGITUDE));
String wrn = memberList.get(i).get(Config.WARNA);

Position destination = Position.fromCoordinates(lng, lat);

getRoute(origin, destination, wrn);
      }
 }
});
```

Gambar 4.5 Kode visualisai rute *dealer*

Hasil luaran dari gambar 4.5 ditampilkan pada gambar 4.6.



Gambar 4.6 Implementasi rute *dealer*

4.1.4 Pencarian *Data*

Dalam aplikasi yang dibuat, pengguna umum maupun anggota dapat melakukan pencarian *dealer*. Kode XML untuk menampilkan menu pencarian yaitu.

```
<android.support.v7.widget.SearchView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/cari_cari"
            android:visibility="gone"
            android:background="@color/white">
```

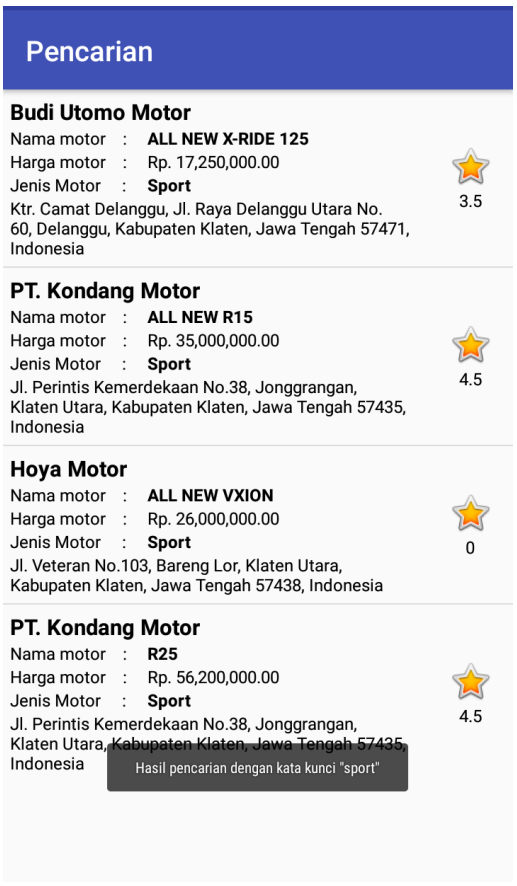
Gambar 4.7 Komponen *SearchView*

Komponen menu pencarian perlu dibuat *adapter* agar bisa ditampilkan di aplikasi. Kode untuk membuat *adapter* menu pencarian yaitu.

```
String[] data = {Config.ID, Config.NAMA_MOTOR,
Config.HARGA,Config.JENIS, Config.NAMA_DEALER,
Config.ALAMAT, Config.LATITUDE,Config.LONGITUDE,
Config.NILAI};

int[] form = {R.id.hasil_id_dealer, R.id.hasil_nama_motor,
R.id.hasil_harga_motor,R.id.hasil_jenis_motor,R.id.hasil_nam
a_dealer,R.id.hasil_alamat_dealer,R.id.hasil_latitude,R.id.h
asli_longitude,R.id.hasil_nilai_dealer};

ListAdapter adapter = new
SimpleAdapter(getApplicationContext(), memberList,
R.layout.list_pencarian,data,form);

listCari.setAdapter(adapter);
}
```

Gambar 4.8 *Adapter* menu pencarian

Hasil luaran dari komponen dan adapter menu pencarian diatas ditampilkan pada gambar 4.9.



Gambar 4.9 Implementasi pencarian *dealer*

4.1.5 Pemberian *Rating Dealer*

Pengguna aplikasi yang akan memberi *rating* harus melakukan *login*. Oleh karena itu, ketika pengguna akan memberikan *rating* pada *dealer* tertentu, harus di periksa status *login* dari pengguna. Adapun kode untuk pemeriksaan *login* dan pemberian *rating* yaitu.

```
@Override
protected Map<String, String> getParams() {
    Map<String, String> params = new HashMap<String,
String>();
SQLiteHandler db = new SQLiteHandler(BeriRating.this);
HashMap<String, String> user = db.getUserDetails();
String idUser = user.get(Config.ID);
     params.put(Config.ID, id);
     params.put(Config.USER, idUser);
     params.put(Config.NILAI_FA, fa);
     params.put(Config.NILAI_KEB, keb);
     params.put(Config.NILAI_WKT, wk);
     return params;
    }
```

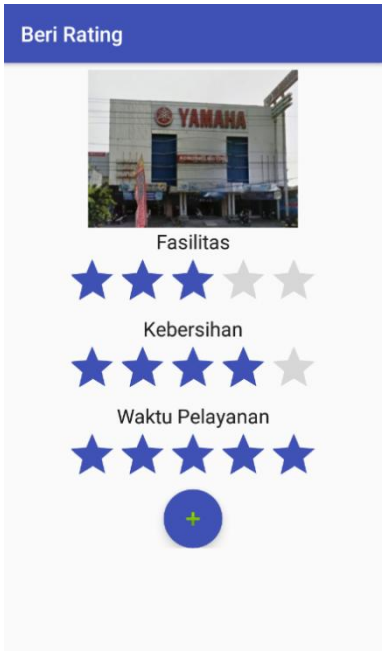
Gambar 4.10 Kirim data *rating* ke *web service*

Layout untuk memberikan rating *dealer* dibuat *form* dengan menggunakan *rating bar*. Kode untuk *layout* penilaian *dealer* yaitu.

```
<RatingBar
android:id="@+id/rating_fasilitas"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center_horizontal"
android:numStars="5"
android:stepSize="1" />
```

Gambar 4.11 Komponen *rating bar* untuk memberi nilai

Tampilan implementasi *layout* untuk pemberian *rating* dengan *rating bar* ditampilkan pada gambar 4.12.



Gambar 4.12 Tampilan dialog pemberian *rating dealer*

4.2 Pembahasan

4.2.1 Pemberian *Rating Dealer*

Pemberian *rating* yang dilakukan anggota, akan disimpan ke *server* sehingga aplikasi harus mengirim data anggota, data *rating*, dan data *dealer* yang akan diberi nilai. Pengguna memberikan nilai kepada *dealer* terntentu dengan menggunakan komponen *rating bar*. Jumlah *rating* yang dapat diberikan pada masing-masing kategori penilaian adalah lima bintang. Kategori yang dapat dinilai oleh anggota adalah fasilitas, kebersihan dan waktu pelayanan *dealer*. *Method* yang digunakan untuk mengirim data adalah *method* POST. Contoh kode untuk mengirim data pada pemberian *rating dealer* yaitu.

```
StringRequest strReq = new
StringRequest(Request.Method.POST,
Config.URL_BERI_NILAI, new Response.Listener<String>() {
@Override
protected Map<String, String> getParams() {
Map<String, String> params = new HashMap<String, String>();
SQLiteHandler db = new SQLiteHandler(BeriRating.this);
HashMap<String, String> user = db.getUserDetails();
String idUser = user.get(Config.ID);
                params.put(Config.ID, id);
                params.put(Config.USER, idUser);
                params.put(Config.NILAI_FA, fa);
                params.put(Config.NILAI_KEB, keb);
                params.put(Config.NILAI_WKT, wk);
                return params;
            }
```

Gambar 4.13 Kode untuk mengirim data *rating dealer*

4.2.2 Visualisasi *Rating Dealer* Pada Peta

Dalam memvisualisasikan *marker* lokasi *dealer*, *dealer* dengan *rating* tertinggi ditampilkan dengan ukuran *marker* yang terbesar sedangkan *dealer* dengan nilai *rating* yang lebih rendah ditampilkan dengan *marker* yang lebih kecil. Perbandingan visualisasi ukuran *marker* berdasarkan rating dealer telah dipaparkan pada tabel 4.1.

Dalam melakukan penyeleksian ukuran *marker* berdasarkan *rating dealer* perlu diambil nilai *rating* semua *dealer* dari *database*. Nilai *rating* dari *dealer* diambil dengan menggunakan teknologi JSON. Untuk mengambil nilai dari masing-masing *dealer* kode yang digunakan adalah.

```
try {
    JSONObject semua = new JSONObject(result);
    JSONArray user = semua.getJSONArray(Config.USER);
    for (int i = 0; i < user.length(); i++) {
        JSONObject detail = user.getJSONObject(i);
        final String nil  = detail.getString(Config.NILAI);
       ...
       }
}
```

Gambar 4.14 Kode untuk mengambil nilai dengan teknologi JSON

Dari variabel nilai yang sudah didapat, kemudian dilakukan seleksi dengan memanggil fungsi yang telah dipaparkan pada gambar 4.4. kode untuk memanggil fungsi pada gambar 4.4 yaitu.

```
icon = getIcon(nil);
```

Variabel *icon* kemudian di atur kedalam *marker* pada *mapview* dengan kode.

```
map.addMarker(new MarkerOptions()
        .position(new LatLng(Double.parseDouble(lat),
Double.parseDouble(lng)))
        .setTitle(nama)
        .setSnippet(alamat)
        .icon(icon)
);
```

Gambar 4.15 Kode visualisasi *marker* berdasrakan *rating*

4.2.3 Menampilkan Rute *Dealer*

Untuk menampilkan rute dari semua *dealer* yang ada, diperlukan data koordinat dari semua *dealer*. Data-data koordinat *dealer* didapatkan dari *database* dengan menggunakan *library Volley*. Dalam *library* ini ada kelas *Volley* yang dapat digunakan untuk menangkap respon dari *server*. Nilai respon yang ditangkap haruslah bebentuk JSON sehingga dapat diolah dengan menggunakan kelas JSONObject. Data-data koordinat yang didapat dari *server* menjadi tujuan akhir rute dan posisi pengguna menjadi titik awal rute. Contoh hasil implementasi dari tampilan visualisasi rute dipaparkan pada gambar 4.5. Pada gambar ini satu posisi pengguna menjadi titik awal rute, sedangkan koordinat-koordinat *dealer* menjadi tujuan akhir rute.

Rute ditampilkan dengan menggunakan bermacam-macam warna agar pengguna dapat membedakan rute untuk menuju *dealer* satu dengan rute menuju

dealer lainnya. Warna dari masing-masing *dealer* di tentukan dengan menggunakan bilangan heksadesimal yang sudah tersimpan pada *database.* Nilai warna yang sudah tersimpan pada *database* kemudian diubah menjadi format warna dengan kode java

```
color(Color.parseColor(color))
```

Dengan diubah kedalam format warna, maka rute yang dibuat akan berwarna sesuai dengan kode heksadesimalnya.

Untuk membuat rute diperlukan sebuah *polyline* atau garis yang menghubungkan antara titik awal dengan titik tujuan. Untuk membuat garis pada peta fungsi yang digunakan adalah.

```
private void getRoute(Position origin, Position destination,
final String color) {
    client = new MapboxDirections.Builder()
                .setOrigin(origin)
                .setDestination(destination)

.setProfile(DirectionsCriteria.PROFILE_DRIVING_TRAFFIC)
                .setAccessToken(Mapbox.getAccessToken())
                .build();
.....
```

Gambar 4.16 Fungsi untuk mendapatkan rute dengan *Direction* SDK

Dengan menggunakan kelas *MapboxDirections* akan diperoleh respon dari server berupa data JSON yang akan membuat titik-titik penghubung untuk membuat rute. Contoh respon dari JSON *direction* pada *Mapbox* yaitu.

Gambar 4.17 Respon JSON dari *Mapbox API*

Dari respon pada gambar 4.13 kemudian diambil data *geometry* untuk membuat titik-titik penghubung antara titik awal dengan titik tujuan. Kode untuk mengambil nilai *geometry* yaitu.

```
LineString lineString =
LineString.fromPolyline(route.getGeometry(), PRECISION_6);
```

4.2.4 Pencarian Data

Pencarian data pada sistem dilakukan dengan mengambil semua data *dealer* pada database *MySQL server*. Untuk mengambil data dari *database server* diperlukan API yang dibuat dengan menggunakan kode PHP. Dengan

menggunakan pemrograman PHP, data dari *database* akan diubah kedalam bentuk JSON. Data perlu diubah kedalam bentuk JSON karena pemrograman *Java* pada *Android* yang digunakan dibuat untuk membaca respon berupa JSON. Kelas yang digunakan pada pemrograman *Android* adalah *Volley*. *Volley* akan menangkap respon dari suatu *Uniform Resouce Locator* (URL) kedalam bentuk *string*.

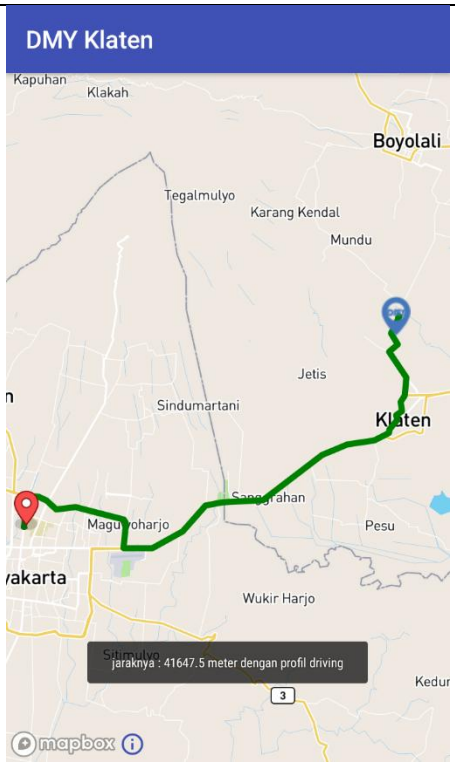
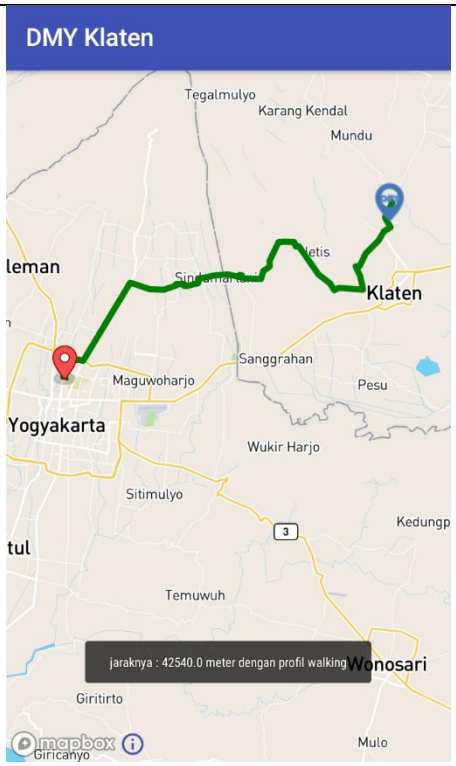
```
getByJenis(new MainActivity.VolleyCallBack() {
    @Override
    public void onSuccess(String result) {
        String alamat = "";
        Log.d("pencarianres: ", result);
```

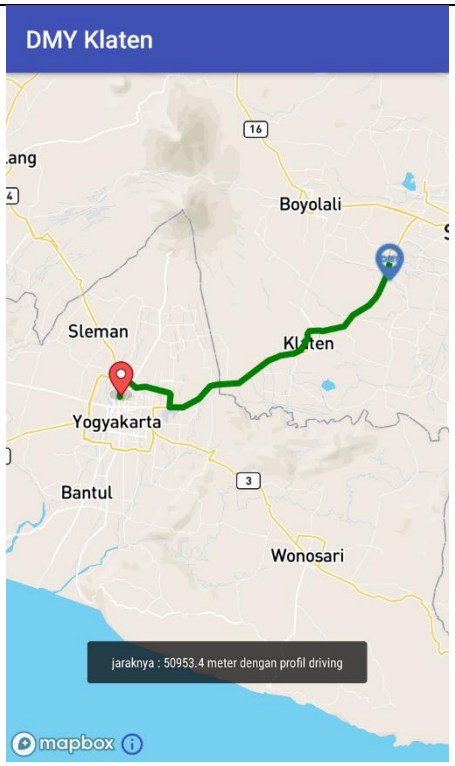
Gambar 4.18 Hasil JSON pada variabel *result*

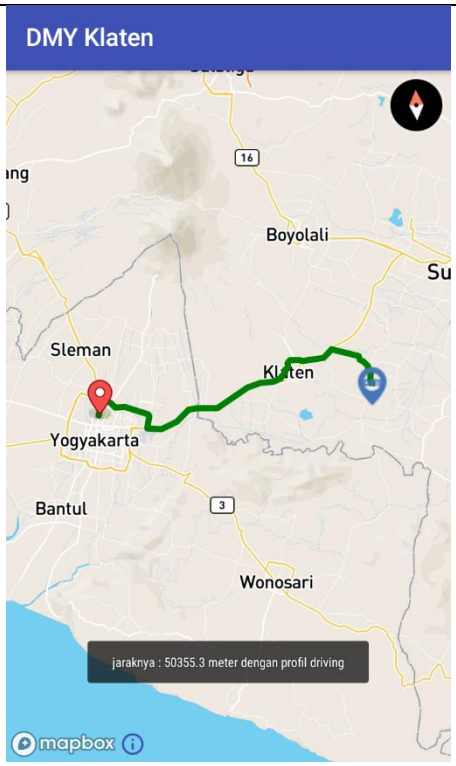
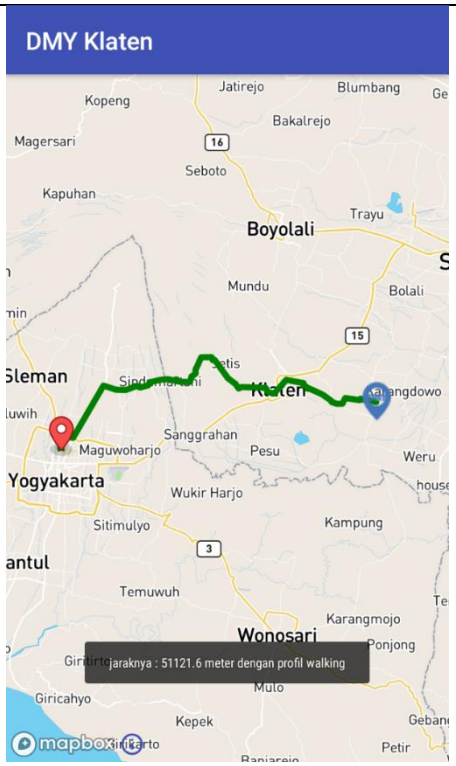
Contoh kode *string* yang diterima oleh kelas *Volley* dipaparkan pada gambar 4.14. *String result* adalah respon yang diterima dari *server*. *String* ini kemudian diproses dengan menggunakan kelas *JSONObject* atau *JSONArray*. Dengan menggunakan kelas *JSONObject* atau *JSONArray* objek ataupun *array* yang ada pada sebuah JSON dapat dibaca dan disimpan pada variabel tertentu. Jika JSON sudah dibaca kemudian disimpan pada variabel, maka dapat dibuat sebuah *adapter* yang akan digunakan untuk mengisi *listview* hasil pencarian.

4.2.5 Uji Coba

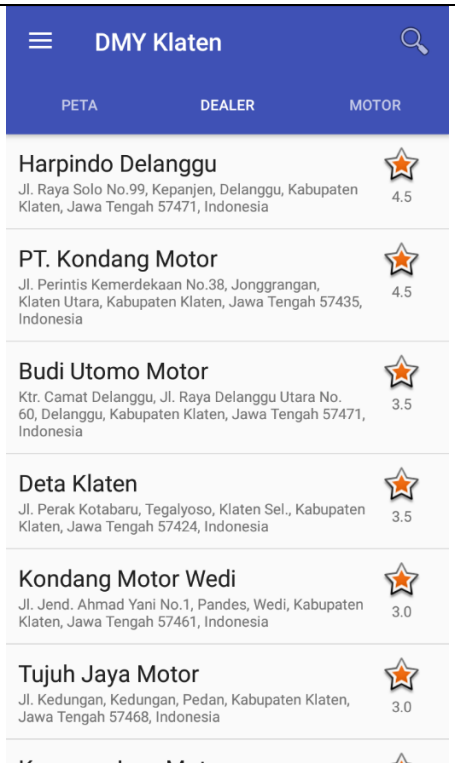
Tabel 4.2 Uji coba rute dengan *direction* SDK

No	Dengan *profile driving*	Dengan *profile walking*
1		
	Dengan menggunakan *profile driving* yang disediakan *mapbox*, jarak menuju kusuma jaya motor adalah 41647 meter	Dengan menggunakan *profile walking* yang disesiakan *mapbox*, jarak menuju kusuma jaya motor adalah 42540 meter
	Jadi, kesimpulan yang dapat ditarik dari pengujian di atas, jarak menggunakan profile driving lebih pendek dibandingkan dengan profile walking, dengan selisih perhitungan jarak 42540 – 41647 = 893 meter. Berdasarkan hasil perhitungan menggunakan fungsi *getDistance()* dari *library direction* SDK. Kemudian untuk rute yang dilalui jika menggunakan profile walking, rute akan disarankan bukan melalui jalan raya melainkan melalui jalur pedesaan.	

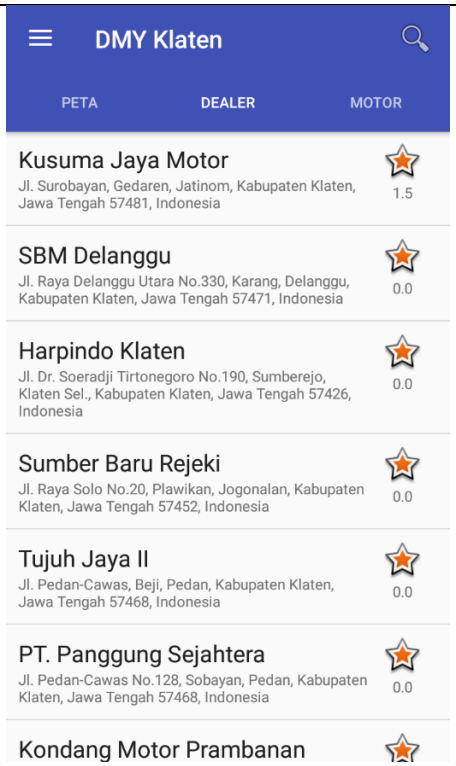
No	Dengan *profile driving*	Dengan *profile walking*
2		
	Dengan menggunakan profile driving yang disediakan mapbox, jarak menuju harpindo delangu adalah 50953 meter	Dengan menggunakan profile driving yang disediakan mapbox, jarak menuju harpindo delangu adalah 64040
	Jadi, kesimpulan yang dapat ditarik dari pengujian di atas, jarak menggunakan profile driving lebih pendek dibandingkan dengan profile walking, dengan selisih perhitungan jarak 64040– 50953 = 13087meter. Berdasarkan hasil perhitungan menggunakan fungsi *getDistance()* dari *library direction* SDK. Kemudian untuk rute yang dilalui jika menggunakan profile walking, rute akan disarankan bukan melalui jalan raya melainkan melalui jalur pedesaan.	

No	Dengan *profile driving*	Dengan *profile walking*
3		
	Dengan menggunakan profile driving yang disediakan mapbox, jarak menuju tujuh jaya II adalah 50353 meter	Dengan menggunakan profile driving yang disediakan mapbox, jarak menuju tujuh jaya II adalah 51121 meter
	Jadi, kesimpulan yang dapat ditarik dari pengujian di atas, jarak menggunakan profile driving lebih pendek dibandingkan dengan profile walking, dengan selisih perhitungan jarak 51121 – 50353 = 768 meter. Berdasarkan hasil perhitungan menggunakan fungsi *getDistance()* dari *library direction* SDK. Kemudian untuk rute yang dilalui jika menggunakan profile walking, rute akan disarankan bukan melalui jalan raya melainkan melalui jalur pedesaan.	

Tabel 4.3 Uji coba data *rating* di visualisaikan pada *marker*

<table>
<tr><th>No</th><th>Nilai Dealer</th><th>Perbandingan marker dealer</th></tr>
<tr><td>1</td><td> 


</td></tr>
<tr><td></td><td colspan="2">Pengujian pengimplementasian marker dealer dengan rating tertinggi ditampilkan dengan marker terbesar. Pada pengujian pertama, dealer Harpindo Delanggu memiliki rating tertinggi yakni 4.5, sehingga dapat terlihat pada peta bahwa marker dari dealer tersebut berukuran paling besar dibandingkan dengan dua dealer lain yang terlihat pada gambar.</td></tr>
</table>

No	Nilai *Dealer*	Perbandingan *marker dealer*
2		
	Pengujian pengimplementasian *marker dealer* dengan *rating* terendah ditampilkan dengan *marker* terkecil. Pada pengujian kedua, *dealer* PT. Panggung Sejahtera memiliki *rating* terendah yakni 0.0, sehingga dapat terlihat pada peta bahwa *marker* dari *dealer* tersebut berukuran paling kecil dibandingkan dengan *dealer* lain yang terlihat pada gambar.	